Exploiting Parallelism in a Scientific Discovery System to Improve Scalability[1]

Gehad Galal and Diane J. Cook

Department of Computer Science and Engineering
University of Texas at Arlington
Box 19015, Arlington, TX 76019
Phone: (817) 272-3606
{galal,cook}@cse.uta.edu

Topic: knowledge discovery, data mining

Abstract

The large amount of data collected today is quickly overwhelming researchers' abilities to interpret the data and discover interesting patterns. Knowledge discovery and data mining approaches hold the potential to automate the interpretation process, but these approaches frequently utilize computationally expensive algorithms. This computational weakness can be overcome by identifying and exploiting parallelism in the discovery system. In this paper, we introduce serial, parallel, and distributed versions of the the SUBDUE discovery system. We analyze the scalability of all versions of SUBDUE. Both the polynomial processing constraints on SUBDUE and parallel algorithms afford considerable scalability for large databases.

1 Introduction

One of the barriers to the integration of scientific discovery methods into practical data mining approaches is their lack of scalability. Many scientific discovery systems are motivated from the desire to evaluate the correctness of a discovery method without regard to the method's scalability.

Another factor is that some scientific discovery systems deal with richer data representations that only degrade scalability. A number of linear, attribute-value-based approaches have been developed that discover concepts in databases and can address issues of data relevance, missing data, noise, and utilization of domain knowledge. However, much of the data being collected is structural, requiring tools for the analysis and discovery of concepts in structural data [3].

In this paper we describe a structure-based approach to discovery in the SUBDUE system. We demonstrate the scalability of the serial version of SUBDUE, and then describe parallel implementations of the system that further improve performance on large databases. Improving the scalability of scientific discovery systems will help break down the barrier excluding these techniques from practical data mining approaches.

2 Overview of SUBDUE

We have developed a method for discovering substructures in databases using the minimum description length principle introduced by Rissanen [4] and embodied in the SUBDUE system. SUBDUE discovers substructures that compress the original data and represent structural concepts in the data. Once a substructure is discovered, the substructure is used to simplify the data by replacing instances of the substructure with a pointer to the newly discovered substructure. The discovered substructures allow abstraction over detailed structures in the original data.

[1]This research supported by NASA grant NAS5-32337 and NSF grant IRI-9502260.

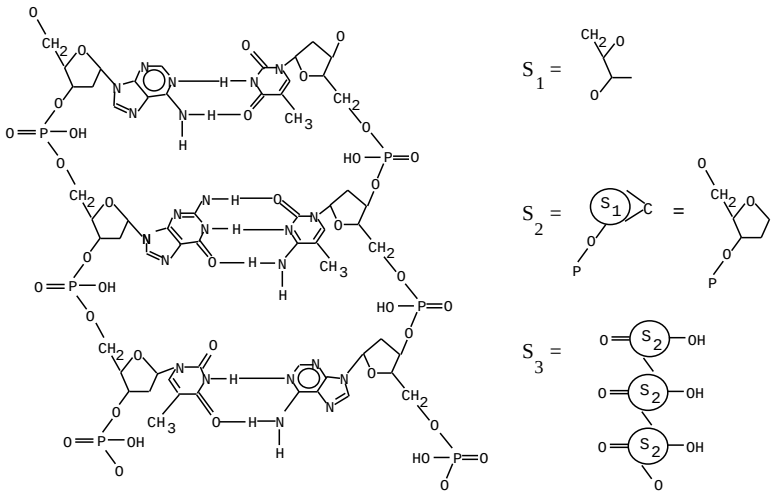

Figure 1: Sample results of Subdue on a protein sequence.

The substructure discovery system represents structural data as a labeled graph. Objects in the data map to vertices or small subgraphs in the graph, and relationships between objects map to directed or undirected edges in the graph. A *substructure* is a connected subgraph within the graphical representation. This graphical representation serves as input to the substructure discovery system. An *instance* of a substructure in an input graph is a set of vertices and edges from the input graph that match the graphical representation of the substructure.

Figure 1 shows a sample input database containing a portion of a DNA sequence. In this case, atoms and small molecules are represented with labeled vertices in the graph, and bonds between atoms are represented with labeled edges in the graph. SUBDUE discovers substructure S_1 from the input database. After compressing the original database using S_1, SUBDUE finds substructure S_2, which when used to compress the database further allows SUBDUE to find substructure S_3. Such repeated application of SUBDUE generates a hierarchical description of the structures in the database.

The substructure discovery algorithm used by SUBDUE is a beam search. The algorithm begins with the substructure matching a single vertex in the graph. Each iteration, the algorithm selects the best substructure and incrementally expands the instances of the substructure. The new unique substructures become candidates for further expansion. The algorithm searches for the best substructure until all possible substructures have been considered or the total amount of computation exceeds a given limit. Evaluation of each substructure is determined by how well the substructure compresses the description length of the database. Because instances of a substructure can appear in different forms throughout the database, an inexact graph match is used to identify substructure instances. SUBDUE has been successfully applied to databases in domains including image analysis, CAD circuit analysis, Chinese character databases, program source code, chemical reaction chains, Brookhaven protein databases, and artificially-generated databases. Evaluation of these applications is described elsewhere [1, 2]. Although the inexact graph match algorithm builds upon a known NP-Complete problem, the user can constrain the search and yield a polynomial-time performance.

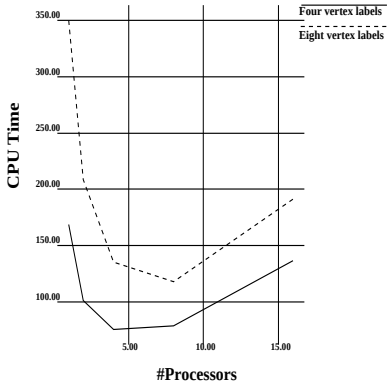

Figure 2: Processing time of P-SUBDUE

3 Parallel and Distributed Approaches

To allow SUBDUE to scale beyond the capabilities of the serial implementation, we have designed and implemented several parallel and distributed versions of SUBDUE that are shown to yield performance improvement over serial SUBDUE.

3.1 Parallel Search SUBDUE

This section describes a parallel approach to SUBDUE that distributes the search for candidate substructures among individual processors. When Parallel-SUBDUE (P-SUBDUE) is run on a graph with M vertex labels using P processors, processor i begins processing a candidate substructure corresponding to the ith unique label in the graph ($i < M$). If $P < M$, the remaining vertex labels are assigned to the last processor. On the other hand, if $P > M$ some processors will initially be idle. Each processor receives a copy of the entire input graph, and begins processing its assigned candidate substructures.

Each processor continues expanding and processing a portion of the possible substructures until the combined number of processed substructures exceeds a given limit or until all processors are idle. Note that there exists a danger of replicating work across multiple processors when substructures are expanded. To prevent this duplication of effort, P-SUBDUE constrains processors expanding a substructure to only include vertices with a label index greater than the processor ID.

When a processor runs out of work, it requests work from a neighboring processor. If the neighbor has a sufficient number of substructure candidates left, the neighbor passes the highest-valued substructure and corresponding instances to the requesting processor. If no work can be shared, the requesting processor continues to ask for work until work can be shared or P-SUBDUE finishes computation.

Quality control is also imposed on the processors in the P-SUBDUE system. One processor is designated as the master processor, and this master regularly receives status information on each processor. Using this information, at regular intervals the master computes the overall average substructure value and sends this information to all processors. Each individual processor then prunes from its list all substructure candidates whose value is less than the global average. In this way no processor is working on a poor substructure candidate that will not likely affect the results of the system.

Figure 2 graphs the processing time using multiple processors on an nCUBE 2 to analyze a database describing the metabolic cycle of citric acid. Note that P-SUBDUE benefits from a large number of processors when there are more unique labels in the input graph. With fewer unique

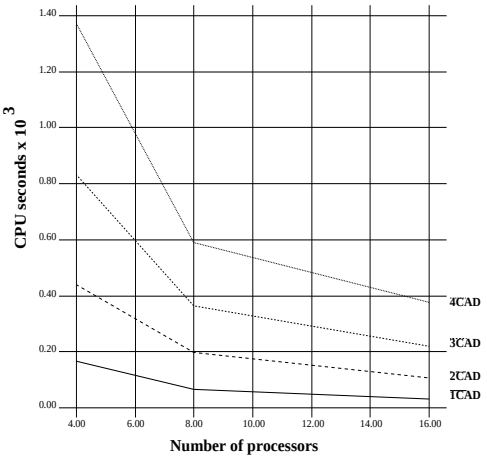

Figure 3: Processing time of PSQ-SUBDUE on CAD database

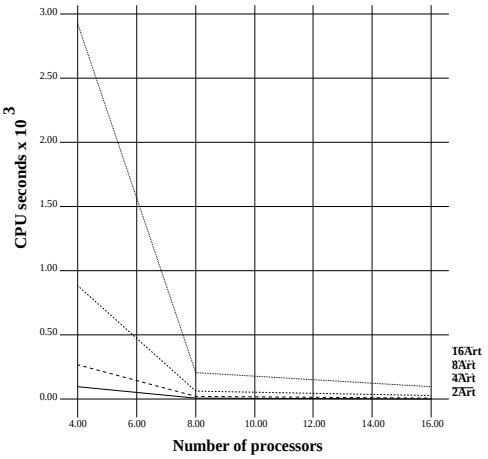

Figure 4: Processing time of PSQ-SUBDUE on Artificial database

input labels, additional overhead results from increased load balancing. We are continuing to refine the distribution and load balancing schemes to achieve even greater speedup from P-SUBDUE.

3.2 Parallel Shared Queue SUBDUE

Each processor executing the Parallel Shared Queue version of SUBDUE, PSQ-SUBDUE, stores the entire input database in local memory, informs the master processor of the results of each expansion step and waits for work to be assigned to them by the master. A single queue of potential substructure definitions is stored on the master processor.

To perform discovery using PSQ-SUBDUE, each processor starts by discovering unique initial substructures (single-vertex substructures) in a manner similar to P-SUBDUE, and expands these substructures by one step in order to obtain substructures large enough to help compress the database. The processors then inform the master processor of the discovered substructure definition, and wait for a response from the master.

The master processor stores and updates the global substructure queue, which represents a queue of the best substructures discovered so far. The master stores only substructure definitions — the instances of each substructure are stored by the processor which discovered the substructure. The master discards substructure definitions if they have previously been considered or are of low value. Otherwise, the substructure is added to the global queue. The master then allocates a new piece of work to the processor. The discovery algorithm terminates when the global search queue is empty or when the number of evaluated substructures exceeds a user-specified threshold. This version of SUBDUE should be expected to outperform P-SUBDUE when a greater number of processors is utilized, because the speedup is not limited to the number of unique vertex labels found in the input graph.

Figures 3 and 4 demonstrate the run-time performance of PSQ-SUBDUE on increasingly-large databases in two separate domains. The first domain consists of a set of graphs representing Computer-Aided Design (CAD) circuits. "CAD1" represents a CAD database with approximately 8500 vertices and 20,000 edges. "CADn" represents a database that is n times as large as CAD1. The second domain consists of artificially-generated graphs. "Art1" represents an artificial graph with 1000 vertices and 2500 edges. "ARTn" represents a database that is n times as large as Art1.

As shown by these results, PSQ-SUBDUE can actually achieve superlinear speedup over serial SUBDUE in terms of number of evaluated substructures. This is due to the difference in search techniques. The sequential version of SUBDUE always chooses the best substructure next for expansion, which causes larger substructures to be evaluated before lesser-valued small substructures. In contrast, PSQ-SUBDUE can process many small substructures before moving on to larger substructures. However, serial SUBDUE will sometimes discover higher-valued substructures.

3.3 Distributed Data SUBDUE

In all of the approaches discussed so far, the entire input graph is expected to fit into the physical memory of a single processor. This expection represents an unrealistic limitation for the large data mining and knowledge discovery applications found in today's world. In contrast, the Distributed Data version of SUBDUE, DD-SUBDUE, distributes the data among a network of processors or workstations thus relaxes this requirement.

DD-SUBDUE must first partition the input graph into n partitions for n processors. The partition should equally balance the work load among the processors. In addition, important edges, or edges that are included in instances of the best overall substructure, should not be cut by the partitioning algorithm. We make use of the Metis system developed at the University of Minnesota to perform graph partitioning.

Each processor executes the sequential version of SUBDUE on its local partition, and broadcasts an abstract description of its results to all other processors. Each processor must evaluate the broadcast substructure definitions on its local partition of the graph and inform the master processor of the evaluation results. The master then determines the overall best substructure. In addition to implementing DD-SUBDUE on the nCUBE 2, we also ported the system to a network of Sun workstations using MPI. To test DD-SUBDUE, we CAD graphs ranging from 8,500 vertices to 2 million vertices, and artificial graphs ranging up to 2 million vertices and 5 million edges.

In addition to superlinear speedup in almost all cases, the quality of the substructures discovered by DD-SUBDUE are in most cases much better than those discovered by the sequential version with the same parameters. The actual run time is small even for the largest database, which required 100 minutes of nCUBE time.

The distributed data version of SUBDUE can offer benefits to the knowledge discovery and data mining community both in terms of run-time performance and space requirements. Future versions of the system will monitor substructure expansions and dynamically repartition the graph if high-valued substructures appear to be found at a partition boundary.

4 Conclusions

The increasing structural component of today's databases requires data mining algorithms to be capable of handling structural information. The SUBDUE system is specifically designed to discover knowledge in structural databases.

In this paper, we analyze the ability of SUBDUE to scale to large databases. The described parallel and distributed implementations of SUBDUE allow us to investigate methods for improving the scalability of scientific discovery systems.

References

[1] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

[2] D. J. Cook, L. B. Holder, and S. Djoko. Scalable discovery of informative structural concepts using domain knowledge. *IEEE Expert*, 11(5), 1996.

[3] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 1, pages 1–34. MIT Press, 1996.

[4] J Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific Publishing Company, 1989.